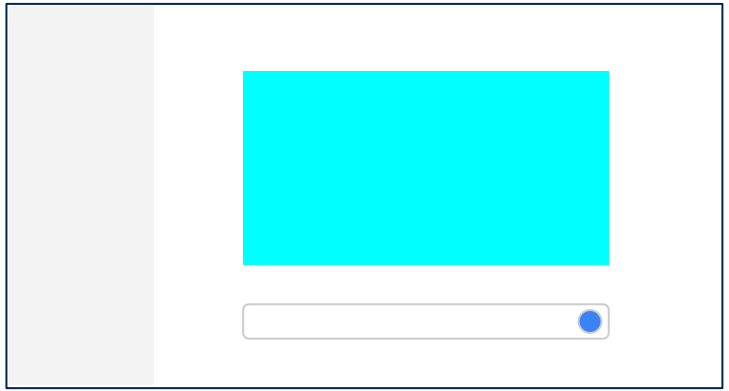
Two Users, One App

OpenAI

Designing MCP Apps for Humans and Agents.

Two new target groups in web development

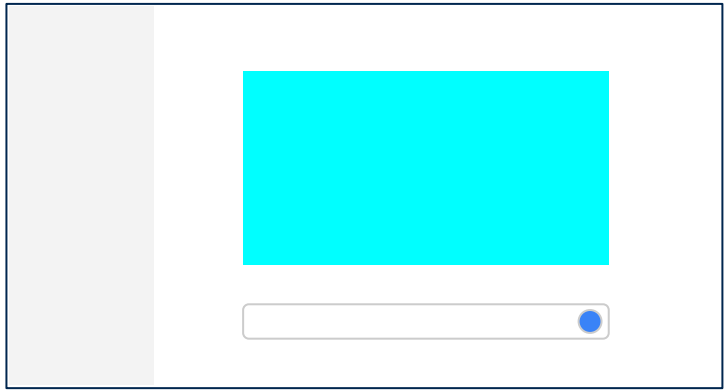


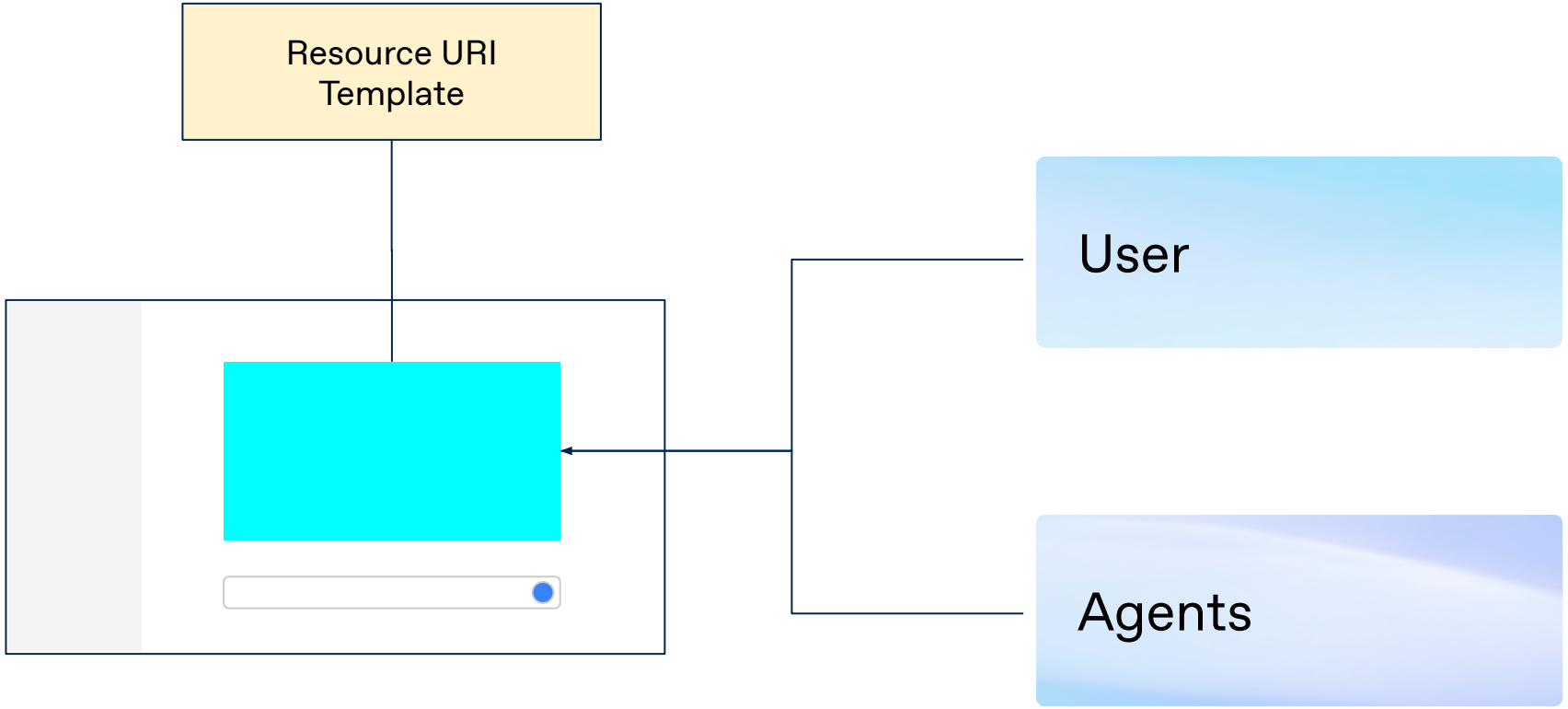
MCP Apps

Computer Use / Browser Use / WebMCP

01 User Experience

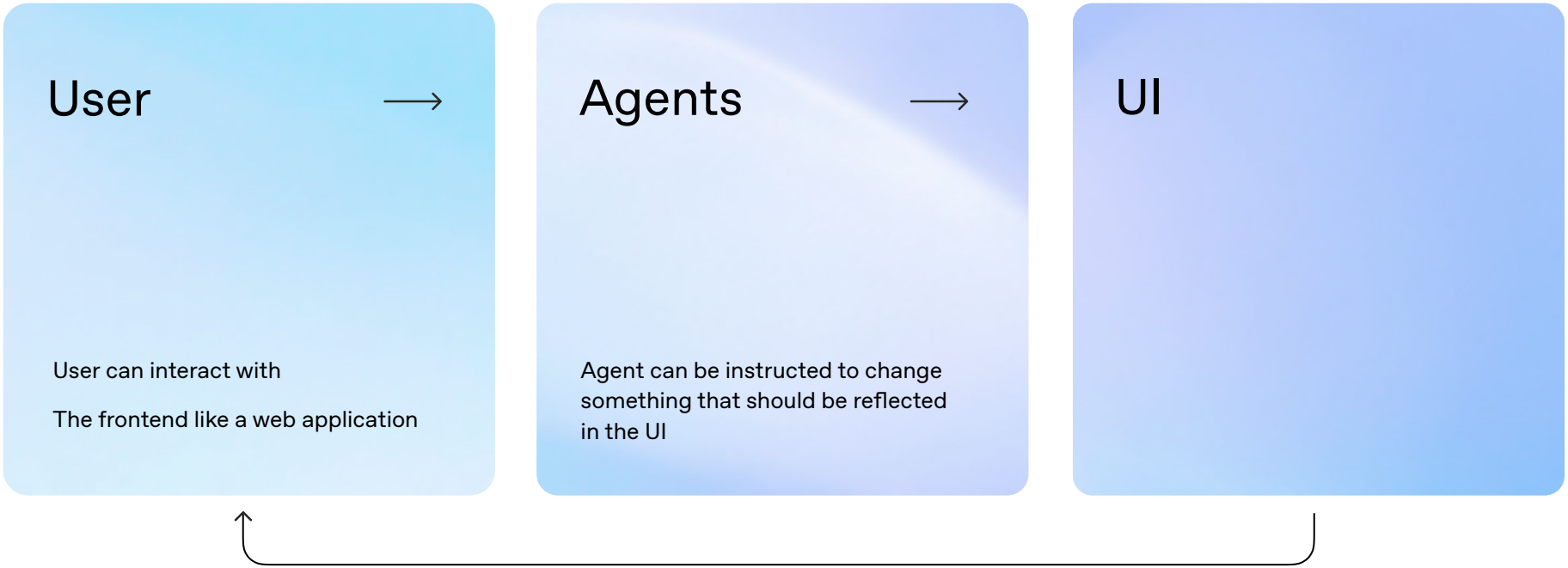
MCP Apps





Demo

Frontend State needs Adaptability from multiple places



1. Shared commands

```
async function select(id, expectedRevision) {
  commitSelection(id, expectedRevision);
  render(state);
  await app.updateModelContext({
    structuredContent: structuredClone(state)
  });
}

const onUserSelect = id => select(id, state.revision);
const onAgentSelect = ({ id, revision }) => select(id, revision);
```

Use the same state management functions.

2. Revision checks

```
function commitSelection(id, expectedRevision) {
  if (state.revision !== expectedRevision) {
    throw new Error("Selection changed");
  }
  if (state.selection === id) return;
  state.selection = id;
  state.revision++;
}

commitSelection("garden", 7);
await app.updateModelContext({
  structuredContent: structuredClone(state)
});
```

The revision guard runs before the write.

3. Separate draft state

```
state.draft = { id: crypto.randomUUID(),
  selection: "garden", base: state.revision };
await app.updateModelContext({
  structuredContent: structuredClone(state)
});

async function accept(id) {
  const draft = state.draft;
  if (!draft || draft.id !== id) throw Error("Draft changed");
  commitSelection(draft.selection, draft.base);
  state.draft = null;
  await app.updateModelContext({
    structuredContent: structuredClone(state)
  });
}
```

The draft stays separate until the user accepts it.

Resources

OpenAI Resources

MCP server and UI quickstart

Add UI — Manage state

Open AI Apps SDK examples

MCP Spec

MCP Apps quickstart

MCP Apps patterns

Thank you

Florian Bauer
Member of Technical Staff
Plugin Developer Ecosystem